(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-533054
(P2004-533054A)
(43)公表日　平成16年10月28日(2004.10.28)

(51)Int.Cl.[7]	F I			テーマコード（参考）
G06F 9/46	G06F	9/46	350	5B098
	G06F	9/46	340A	

審査請求 未請求　予備審査請求 有　（全 43 頁）

(21)出願番号	特願2002-589978 (P2002-589978)
(86)(22)出願日	平成14年5月15日 (2002.5.15)
(85)翻訳文提出日	平成15年11月14日 (2003.11.14)
(86)国際出願番号	PCT/US2002/015378
(87)国際公開番号	WO2002/093369
(87)国際公開日	平成14年11月21日 (2002.11.21)
(31)優先権主張番号	09/859,209
(32)優先日	平成13年5月16日 (2001.5.16)
(33)優先権主張国	米国 (US)
(81)指定国	EP (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW

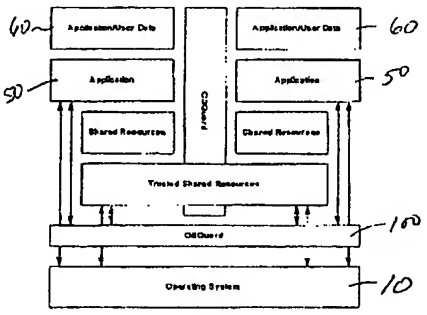
(71)出願人　501153496
ソフトリシティ　インコーポレイテッド
アメリカ合衆国　０２１１０　マサチューセッツ州　ボストン　コングレス　ストリート　３３２
(74)代理人　100083932
弁理士　廣江　武典
(74)代理人　100121429
弁理士　宇野　健一
(72)発明者　シェーファー，ステュワート
アメリカ合衆国　マサチューセッツ州　０１９４５，マーブルヘッド，ワン　ガリソン　アベニュー，無番地

Fターム(参考)　5B098 GA02　HH01

最終頁に続く

(54)【発明の名称】オペレーティングシステム抽象化／保護レイヤ

(57)【要約】
本発明は、クライアントコンピュータのオペレーティングシステムを変更することなくアプリケーションソフトウェア環境を作成するためのシステムを提供する。このシステムは、オペレーティングシステム抽象化／保護レイヤから成り、抽象化／保護レイヤは実行されているソフトウェアアプリケーションとオペレーティングシステムとの間に配置される。これにより、アプリケーションを実行できる仮想環境が提供され、アプリケーションレベル間の相互作用が実質的に除去される。好ましくは、オペレーティングシステムに対する直接的な変更を、実行されているアプリケーションのコンテキスト内で選択的に行い、抽象化／保護レイヤが管理設定に従って仮想環境を動的に変更する。また、一部の実施形態では、システムが共有システムリソースの使用状況を継続的に監視し、システムコンポーネントに対する変更を適用および除去するサービスとして機能する。このため、本発明は「オペレーティングシステムガード」を定義するものである。これらのコンポーネントには、ＤＬＬおよびその他の共有ライブラリコードで必要とされるセマンティ



【特許請求の範囲】
【請求項１】
クライアントコンピュータのオペレーティングシステムを変更することなくアプリケーションソフトウェア環境を作成するための、オペレーティングシステム抽象化／保護レイヤから成るシステムであって、実行されているソフトウェアアプリケーションと前記オペレーティングシステムとの間に前記抽象化／保護レイヤを挿入するようにし、これにより、アプリケーションを実行できる仮想環境を提供しアプリケーションレベル間の相互作用を実質的に除去するものであることを特徴とするシステム。
【請求項２】
オペレーティングシステムに対する直接的な変更が、実行されているアプリケーションのコンテキスト内で選択的に行われるものであることを特徴とする請求項１記載のシステム。
【請求項３】
抽象化／保護レイヤが管理設定に従って仮想環境を動的に変更するものであることを特徴とする請求項２記載のシステム。
【請求項４】
システムが共有システムリソースの使用状況を継続的に監視し、システムコンポーネントに対する変更を適用および除去するサービスとして機能するものであることを特徴とする請求項１記載のシステム。
【請求項５】
オペレーティングシステムがＷｉｎｄｏｗｓ（登録商標）ベースのオペレーティングシステムであるようにし、Ｗｉｎｄｏｗｓ（登録商標）　Ｒｅｇｉｓｔｒｙと．ｉｎｉファイルに対する操作はすべてＷｉｎ３２　ＡＰＩによって行われるものであり、関数が呼び出されるたびに別の関数またはアプリケーションがその呼び出しを中断するように前記関数をフックするための手段からさらに有することを特徴とする請求項１記載のシステム。
【請求項６】
サーバーから実行されているアプリケーションによる要求であれ、アクティブに管理されている構成キーに対してアプリケーションが出した要求であれ、要求に応えるためにシステムが該当するＡＰＩ関数それぞれをフックするものであることを特徴とする請求項５記載のシステム。
【請求項７】
前記オペレーティングシステム抽象化／保護レイヤは、実行されているアプリケーションインスタンスの数を認識することで、１つのアプリケーションの多重のインスタンスの統合を管理するものであることを特徴とする請求項１記載のシステム。
【請求項８】
実行されているアプリケーションインスタンスが１つのみではない場合、前記オペレーティングシステム抽象化／保護レイヤは、起動時とシャットダウン時の変更を避けるものであることを特徴とする請求項７記載のシステム。
【請求項９】
前記オペレーティングシステム抽象化／保護レイヤが、インストールを実行しないでもインストール環境のように出現する環境をアプリケーションに提供することにより、アプリケーションの実行時にすべての設定が仮想環境にもたらされる「擬似インストール」が作成されるものであることを特徴とする請求項１記載のシステム。
【請求項１０】
クライアントコンピュータ上の情報によってアプリケーションの動作が干渉されたり変更されることを防止するための手段をさらに有することを特徴とする請求項９記載のシステム。
【請求項１１】
管理設定に従って仮想環境を動的に変更するための手段をさらに有することを特徴とする請求項９記載のシステム。

【請求項１２】
単一ソフトウェアアプリケーションの複数のインスタンスが同じクライアントコンピュータ上で実行されるものであり、前記複数のインスタンスの各々が１つの異なるデータベースに接続されるものであることを特徴とする請求項９記載のシステム。
【請求項１３】
単一アプリケーションの前記の複数のインスタンスのうち少なくとも２つが、１つ以上の仮想設定を共有するような、共有・制御されたコンテキストを提供するものであることを特徴とする請求項１２記載のシステム。
【請求項１４】
特定のアプリケーションについてインストール時に命令を受け取るデバイスドライバモニターをさらに有することを特徴とする請求項１記載のシステム。
【請求項１５】
全機能レジストリをアプリケーションに提供するが、基本システムレジストリに対する変更は防止する仮想Ｗｉｎｄｏｗｓ（登録商標）　Ｒｅｇｉｓｔｒｙコンポーネントをさらに有することを特徴とする請求項１記載のシステム。
【請求項１６】
キーおよびキーの値がオペレーティングシステム抽象化／保護レイヤに格納されている場合は、オペレーティングシステム抽象化／保護レイヤが前記キーおよびキーの値で応答し、格納されていない場合は、オペレーティングシステム抽象化／保護レイヤが要求をＷｉｎｄｏｗｓ（登録商標）　Ｒｅｇｉｓｔｒｙまで通過させることを許容するものであることを特徴とする請求項１記載のシステム。
【請求項１７】
前記キーの値の変更が試みられた場合、オペレーティングシステム抽象化／保護レイヤは、それ自身に対する変更のみを許容するものであることを特徴とする請求項１６記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本出願は米国特許出願第０９／４５６，１８１号の一部継続出願であり、その全体が十分に記載されているものとし、引用参考文献としてここに包含される。
【０００２】
本発明は、コンピュータソフトウェア、より詳細にはオペレーティングシステムソフトウェアに関する。
【背景技術】
【０００３】
多くの環境において、特にアプリケーションがネットワークを介して配布されるような環境において最も重要な機能は、複雑なインストール作業なしにアプリケーションをすぐに実行できることである。一般に、一部の従来システムでは、あたかもプログラムがインストールされているかのようにクライアントシステムを変更したり、あるいはソフトウェア自体を実際にインストールした後でこれらの変更を取り消して元の構成を復元する際に、大変手間がかかった。
【発明の開示】
【発明が解決しようとする課題】
【０００４】
このような作業を行う場合、次のような多くの問題が生じる。すなわち、アプリケーションと現在のコンピュータ構成との間に矛盾、不一致があること、同じまたは異なるアプリケーションのインスタンスが複数存在すること、取り消し処理が複雑であるため、アプリケーションに対して厳密な処理を施して、アプリケーションに対するすべての変更の原因を明らかにする必要があること、複数のアプリケーションで共有ファイルと共有システムコンポーネントが使用されているため、取り消し処理とインストール処理が複雑になると

いった問題である。

【課題を解決するための手段】

【０００５】

本発明は、クライアントコンピュータのオペレーティングシステムを変更することなくアプリケーションソフトウェア環境を作成するためのシステムを提供する。このシステムは、オペレーティングシステム抽象化／保護レイヤから成り、抽象化／保護レイヤは実行されているソフトウェアアプリケーションとオペレーティングシステムとの間に配置される。これにより、アプリケーションを実行できる仮想環境が提供され、アプリケーションレベル間の相互作用が実質的に除去される。好ましくは、オペレーティングシステムに対する直接的な変更を、実行されているアプリケーションのコンテキスト内で選択的に行い、抽象化／保護レイヤが管理設定に従って仮想環境を動的に変更する。また、幾つかの実施形態では、システムが共有システムリソースの使用状況を継続的に監視し、システムコンポーネントに対する変更を適用および除去するサービスとして機能する。

【０００６】

したがって、たとえばＷｉｎｄｏｗｓ（登録商標）ベースのオペレーティングシステム内の実施形態では、Ｗｉｎｄｏｗｓ（登録商標）　Ｒｅｇｉｓｔｒｙに対する操作はすべてＷｉｎ３２　ＡＰＩによって行われ、好ましくはシステムが関数をフックするための手段を提供する。これにより、関数が呼び出されるたびに別の関数またはアプリケーションがその呼び出しを中断する。最も好ましくは、サーバーから実行されているアプリケーションによる要求であれ、アクティブに管理されている構成キーに対してアプリケーションが出した要求であれ、要求に応えるためにシステムが適切なＡＰＩ関数それぞれをフックする。

【０００７】

本発明の別の好ましい実施形態では、追加機能が提供される。たとえば、オペレーティングシステム抽象化／保護レイヤが、実行されているアプリケーションインスタンスの数を認識することで、１つのアプリケーションの複数インスタンスの統合を管理する。このような実施形態では、実行されているアプリケーションインスタンスが１つのみではない場合、起動時とシャットダウン時の変更を避けることが最も好ましい。この実施形態では、種々異なるユーザーのために多重のアプリケーションインスタンスを実行できる、マルチユーザーオペレーティングシステムをサポートすることも可能である。

【０００８】

このため、オペレーティングシステム抽象化／保護レイヤが、インストールを実行しないでもインストール環境であるように現れる環境をアプリケーションに提供する。これにより、アプリケーションの実行時にすべての設定が仮想環境にもたらされる「擬似インストール」が作成される。また、アプリケーションがインストールされている場合は、実行時にアプリケーションの動作を動的に変更する。好ましい実施形態は、クライアントコンピュータ上の情報によってアプリケーション動作が干渉障害を受けたり、変更されることを防止するための手段を提供し、最も好ましくは、管理設定に従って仮想環境を動的に変更するための手段を提供する。上記のように、一部の実施形態では、同じクライアントコンピュータ上で単一ソフトウェアアプリケーションの複数のインスタンスを実行することができる（本来そのように作成されていない場合でも）。このような実施形態では、単一アプリケーションの前記の複数のインスタンスのうちの少なくとも２つが１つ以上の仮想設定を共有するような、共有・制御されたコンテキストが提供される。

【発明を実施するための最良の形態】

【０００９】

図１は、本発明、オペレーティングシステム、およびソフトウェアアプリケーションの相対的関係を示すブロック略図である。本発明の好ましい実施形態は、オペレーティングシステム抽象化／保護レイヤ（１００）を提供する。これを「オペレーティングシステムガード」と呼ぶ。内部的には、多数のオペレーティングシステム（１０）が、実行時にアプリケーションが互いに影響し合うことを防止する障害ドメインを備える。しかし、共有さ

れているシステムリソースとその他の多くのオペレーティングシステム機能によって、この保護ドメインが危険にさらされる。オペレーティングシステム抽象化／保護レイヤ（１００）は、アプリケーションレベル間のほとんどの相互作用をなくすために、プログラム制御された障壁をアプリケーション（５０）間に追加する。アプリケーション（５０）とオペレーティングシステム（１０）との間に配置されたアプリケーション抽象化／保護レイヤ（１００）が、オペレーティングシステム（１０）に対する直接的な変更か、実行されているアプリケーションのコンテキスト内に変更を含めるかを選択的に許可する。一例としてＷｉｎｄｏｗｓ（登録商標）ベースのシステムでは通常、Ｗｉｎｄｏｗｓ（登録商標）　Ｒｅｇｉｓｔｒｙに対する操作はすべてＷｉｎ３２　ＡＰＩによって実行される。後述するように、ＱｕｅｒｙＲｅｇＥｘやＧｅｔＰｒｏｆｉｌｅＳｔｒｉｎｇなどのシステム関数が呼び出されるたびに別の関数またはアプリケーションがこの呼び出しを中断するように、これらのシステム関数をフックすることができる。アクティブに管理されているアプリケーションによる要求であれ、アクティブに管理されている構成項目に対してアプリケーションが出した要求であれ、要求に応えるために本発明のオペレーティングシステムガード（１００）が適切なＡＰＩ関数それぞれをフックする。この方法では、これを実行するように明示的に構成されていない場合、本発明によって、エンドユーザーシステムを実際に変更することなくアプリケーション環境を作成することができる。また、実行時にアプリケーションによって実行された変更を存続させたり除去することも容易である。

【００１０】

ここで使用している「オペレーティングシステムガード」という用語は、ターゲットコンピュータまたはクライアントコンピュータ上の、実行されているアプリケーションとオペレーティングシステムとの間に配置され、アプリケーションを実行できる仮想環境を提供するレイヤを意味する。この仮想環境にはいくつかの目的がある。第１の目的は、実行されているアプリケーションによってクライアントコンピュータが変更されることを防止することである。アプリケーションがクライアントコンピュータの基本オペレーティングシステム設定の変更を試みると、この設定は保護され、仮想環境でのみ変更が「行われる」。たとえば、アプリケーションがＭＳＶＣＲＴ．ＤＬＬなどの共有オブジェクトのバージョンを変更しようとすると、この変更がアプリケーションだけにとどめられ、クライアントコンピュータに常駐するコードは影響を受けない。

【００１１】

第２の目的は、インストールを実行しないでもインストール環境であるようにみえる環境（すなわち「擬似インストール」または「インストールのような」環境）を、実行されているアプリケーションに提供することである。アプリケーションの実行時、またはアプリケーションが特定の設定を必要としているちょうどのタイミングですべての設定が仮想環境にもたらされる。たとえば、Ａｄｏｂｅ　Ｐｈｏｔｏｓｈｏｐ（Ｒ）などのコンピュータプログラムが、一連のＷｉｎｄｏｗｓ（登録商標）　ＲｅｇｉｓｔｒｙエントリがＨＫＥＹ＿ＬＯＣＡＬ＿ＭＡＣＨＩＮＥ¥Ｓｏｆｔｗａｒｅ¥Ａｄｏｂｅにあると想定している場合に、Ｐｈｏｔｏｓｈｏｐがインストールされていないためクライアントコンピュータ上のＨＫＥＹ＿ＬＯＣＡＬ＿ＭＡＣＨＩＮＥ¥Ｓｏｆｔｗａｒｅ¥Ａｄｏｂｅにこれらのエントリが存在しなければ、本発明のこの態様に従うシステムは、あたかもクライアントコンピュータ上に常駐するかのように、これらのレジストリエントリをＰｈｏｔｏｓｈｏｐプログラミングコードに対して「示す」。

【００１２】

次に、本発明は、クライアント／ユーザーマシン上に存在する可能性がある情報によってアプリケーションの動作が干渉または変更されることを防止する。たとえば、ユーザーがＨＫＥＹ＿ＬＯＣＡＬ＿ＭＡＣＨＩＮＥ¥Ｓｏｆｔｗａｒｅ¥Ａｄｏｂｅの下にＰｈｏｔｏｓｈｏｐの古いバージョン用のレジストリエントリが既に存在している状態で、新しいバージョンを稼動したい場合、これらのエントリを新しいアプリケーションから隠して競合を防ぐことができる。

【００１３】
最後に、本発明は、アプリケーションが現在書き込まれているため存在しない可能性があるアプリケーション動作をアンロックする。このアンロックが可能なのは、管理設定に従って仮想環境を動的に変更できるからである。たとえば、企業向けソフトウェアアプリケーションの典型的な例では、クライアントアプリケーションは、ユーザーがレジストリ内の設定から接続する必要があるデータベースのアドレスについての設定を読み取ることを期待することがある。このレジストリキーはＨＫＥＹ＿ＬＯＣＡＬ＿ＭＡＣＨＩＮＥに格納されることが多いため、この設定はクライアントコンピュータ全体に適用される。ユーザーが複数のデータベースに接続するには、クライアントを再インストールするか、またはこのレジストリキーの変更方法を知ってアプリケーションを実行するたびにレジストリキーを変更しなければならない。しかし、本発明を実現する場合、アプリケーションの２つのインスタンスを同じクライアントコンピュータ上で実行し、それぞれのインスタンスを異なるデータベースに接続できるようになる。
【００１４】
コンテキスト
この機能があるため、システム内のプライベートコンテキストで各アプリケーションを実行することができる。アプリケーションに対してシステムがどのように見えるか、およびその挙動特性についてプライベートビューが与えられる。本発明は、その固有の性質によってこれを実現する。図２では、２つの個別アプリケーション（５２と５４）または同じアプリケーション（図１の５０）の２つのインスタンスにプライベートコンテキストを提供することができる。このプライベートコンテキストでは、これらのアプリケーションまたはインスタンスが、システムサービス、構成、およびデータの個別コピーまたは異なるコピーを持つようにみえる。好ましい実施態様では、これがシステムの既定動作である。
【００１５】
この着想を拡張することで、本発明のオペレーティングシステムガード（１００）は、複数のアプリケーション（５２と５４）が一部またはすべての仮想設定を共有できるような、共有・制御されたコンテキストを提供することもできる。このことは、Ｍｉｃｒｏｓｏｆｔ　Ｏｆｆｉｃｅなどのアプリケーションスイートや、別のアプリケーションが存在すると動作が異なってくるアプリケーションにとって重要である。たとえば、メールマージや文書作成機能を実行するためのエンジンとしてＭｉｃｒｏｓｏｆｔ　Ｗｏｒｄが多くのアプリケーションで使用されている。このようなアプリケーションはＷｏｒｄのインストールまたは有無に関する情報を知り、Ｗｏｒｄの機能を利用できなければならない。好ましい実施形態では、同じアプリケーションの２つのインスタンスが既定で単一のコンテキストを共有し、一方、２つの別個のアプリケーションがプライベートコンテキストを維持する。図３に示したように、２つのアプリケーション（５２と５４）を実行可能であり、オペレーティングシステムガード（１００）が、利用可能なシステムリソースの共有ビューを提供する。
【００１６】
設計
図４に示したように、オペレーティングシステムガードは次のサブシステムから構成される。すなわち、コア（１０２）、構成マネージャ（１０４）、ファイルマネージャ（１０６）、共有オブジェクトマネージャ（１０８）、デバイスマネージャ（１１０）、フォントマネージャ（１１２）、プロセスマネージャ（１２０）、プロセス環境マネージャ（１１４）、ローダー（１１６）、およびリカバリマネージャ（１１８）から構成される。コア（１０２）、プロセスマネージャ（１２０）、およびローダー（１１６）を除くすべてのサブシステムは、以下で詳述する仮想化システムの要素である。コア（１０２）は主として、アプリケーションと、構成ファイルで定義されるアプリケーションのコンテキストを管理する。
【００１７】
オペレーティングシステムガードのプロセスマネージャ（１２０）によって、当該のプロ

セスまたはスレッドイベントをコア（１０２）に通知できるようになる。また、プロセスマネージャ（１２０）は、プロセス空間の管理とスレッド処理のために、オペレーティングシステムに依存する実現化に対して抽象化レイヤを提供する。プロセスをまとめてアプリケーションバンドルにすることができる。アプリケーションバンドルとは、すべてが互いに仮想リソースを共有するプロセスのグループである。たとえば、Ｍｉｃｒｏｓｏｆｔ ＷｏｒｄとＭｉｃｒｏｓｏｆｔ Ｅｘｅｃｌがアプリケーション一式として協働するために、仮想レジストリと仮想ファイルシステムを共有する場合がある。プロセスマネージャ（１２０）は、これらのアプリケーションバンドルを「アプリケーション」とみなす。プロセスマネージャ（１２０）がアプリケーションの解放を指示されるまで、アプリケーションに関する情報が存在する。別のプロセスをアプリケーションバンドルにロードする必要がある場合、アプリケーションが解放されていない限りロードが可能である。

【００１８】

本発明のローダーサブシステム（１１６）を使用すれば、実行されているシステムに仮想環境を転送したり、システムから仮想環境を転送することができる。ローダー（１１６）。仮想化サブシステムそれぞれは、ローダー（１１６）に対して構成をシリアル化し、リバースプロセスを通じて構成を取得することができる。さらに、ローダー（１１６）は段階的ロード／アンロードを実行し、個々の段階の結果を組み合わせて単一の環境記述にすることができる。

【００１９】

レジストリと構成

アプリケーションを正しく動作させるには、様々な量の構成情報が必要である。アプリケーションが構成を読み取ることができる構成レコードの数は、ゼロから数千の範囲におよぶ。Ｗｉｎｄｏｗｓ（登録商標）では、構成情報が一般に格納される場所が２つある。すなわち、Ｗｉｎｄｏｗｓ（登録商標） Ｒｅｇｉｓｔｒｙとシステムレベル初期化ファイル（ｗｉｎ．ｉｎｉおよびｓｙｓｔｅｍ．ｉｎｉ）である。さらに、￥ＷＩＮＤＯＷＳ（登録商標）￥ＳＹＳＴＥＭディレクトリは、アプリケーションがアプリケーション固有の構成または初期化ファイルをよく書き込む場所である。また、アプリケーションは、ローカルアプリケーションディレクトリ内の構成ファイルまたはデータファイルを使用して、追加情報を格納する。この情報が独自仕様の形式であるため扱いにくいことが多い。Ｗｉｎｄｏｗｓ（登録商標）以外のプラットフォームにはＲｅｇｉｓｔｒｙに相当するものはないが、構成情報がよく格納されるディレクトリが存在する。Ｘ Ｗｉｎｄｏｗｓ（登録商標）には、ａｐｐ－ｄｅｆａｕｌｔｓディレクトリがある。ＭａｃｉｎｔｏｓｈにはＳｙｓｔｅｍ Ｆｏｌｄｅｒがあり、その他のオペレーティングシステムには対応する要素が存在する。また図２に示したように、ほとんどのＵＮＩＸ（登録商標）システムでは、個々のアプリケーション（５２と５４）それぞれがその構成（１５２と１５４）をローカルに格納することがほとんどである。

【００２０】

本発明の一実施態様には、仮想Ｗｉｎｄｏｗｓ（登録商標） Ｒｅｇｉｓｔｒｙコンポーネントが含まれる。このコンポーネントは全機能レジストリをアプリケーションに提供するが、基本システムレジストリに対する変更は防止する。アプリケーションがアクセスすることを期待するキーはすべて存在するが、これらのキーは仮想レジストリにのみ存在することができる。この方法では、本発明のオペレーティングシステムガード（１００）とＷｉｎｄｏｗｓ（登録商標） Ｒｅｇｉｓｔｒｙが、レジストリにアクセスするための２段階プロセスを形成する。アプリケーションがキーへのアクセスを必要とする場合、アプリケーションはＲｅｇｉｓｔｒｙに問い合わせる。オペレーティングシステムガードは、キーおよびキーの値（値を知っている場合）で応答する。そうでない場合、オペレーティングシステムガードは要求をＷｉｎｄｏｗｓ（登録商標） Ｒｅｇｉｓｔｒｙまで通過させることを許可する。値の変更が試みられた場合、オペレーティングシステムガードは、それ自身に対する変更のみ許可する。次回アプリケーションがキーにアクセスするとき、そのキーはオペレーティングシステムガード内に存在し、要求は実際のＲｅｇｉｓｔｒｙ

まで流れ込まず、変更されない。

【００２１】

オペレーティングシステムガードが使用するキーは、３つの個別セクションで指定される。これらのオペレーティングシステムガードキーはこれらのセクションで、既存キーの変更、キーの存在の削除、またはレジストリへの新しいキーの追加のためのコマンドとして指定される。このようにして、仮想レジストリは、全くシステムの意図通りに出現することができる。このことは、キーの有無がキーの実際の値と同じ程度に重要になることができるため、重要である。

【００２２】

好ましい実施態様では、オペレーティングシステムガードはまず、アプリケーションに対する基本レジストリエントリを含むデータファイルをロードする。次に、ユーザーの基本設定を含む第２のデータファイルがロードされる。最後に、オペレーティングシステムガードは、ユーザーが上書きを禁止されているポリシー項目を含む一連のキーを任意にロードできる。この３つのファイルは順にロードされ、各ファイル内に重複する項目がある場合、前のファイル内の項目が上書きされる。ユーザーが初めてアプリケーションを実行するとき、ユーザー固有の情報が存在せず、アプリケーション既定情報のみ存在するため、第２のデータファイルは存在しない。しかし、各セッションの後、オペレーティングシステムガードはユーザーの変更内容を保存し、将来のセッションで使用するために第２のデータファイルを生成する。

【００２３】

構成ファイルを２つの方法で変更できる。第１の方法では、構成ファイルをアプリケーションによって直接編集できる。この方法では、後述するオペレーティングシステムガードのファイルサブシステムがファイルへの変更を扱う。好ましい実施形態である第２の方法では、アプリケーションがＷｉｎｄｏｗｓ（登録商標）　ＡＰＩ関数ＧｅｔＰｒｏｆｉｌｅＳｔｒｉｎｇやＷｒｉｔｅＰｒｏｆｉｌｅＳｔｒｉｎｇなどを呼び出して、これらのファイルを変更することができる。この方法では、本発明のオペレーティングシステムガードが上述のとおりこれらの呼び出しを横取りし、内部からこれらの呼び出しを実行する。

【００２４】

共有オブジェクト

オペレーティングシステムおよび実行されているアプリケーションで使用される多くのコンポーネントは、複数のアプリケーションまたはインスタンスで共有される。一般に、これは優れた方法である。この方法では、ディスク容量を節約でき、同じファイルを数多くコピーする必要がない。また、頻繁に使用されるコードのライブラリをオペレーティングシステムベンダーとサードパーティが作成・配布できるようになる。Ｗｉｎｄｏｗｓ（登録商標）プラットフォームでは、ＤＬＬ（ダイナミックリンクライブラリ）がアプリケーション内や複数のアプリケーション間で共有されることが多い。その他のプラットフォームでも、問題は同じである。Ｍａｃｉｎｔｏｓｈでは、ＩＮＩＴおよびその他のシステムコンポーネントがアプリケーション用にロードされる。これらのコンポーネントには複数のバージョンが存在し、一度に使用されるのがそれらのうちの１つのみである場合がある。ＵＮＩＸ（登録商標）システムでは、ロード時間の短縮やディスク容量の節約などの理由で動的共有オブジェクト（たとえば”．ｓｏ”ライブラリファイル）がアプリケーションで使用される。多くのプログラムでは既定の”ｌｉｂｃ．ｓｏ”が使用されるが、このライブラリファイルはｌｉｂｃ．ｓｏ．３など一部のバージョンへのシンボリックリンクであることが多い。実際には、この機能が大きな混乱を招いてきた。これらの共有コンポーネントは改訂されることが多く、同じコンポーネントの複数のバージョンがインストールされる。アプリケーション作成者は、作成したソフトウェアが共有コンポーネントの１つまたは一部のバージョンとのみ動作できることを知っている。このため実際には、アプリケーションは希望のバージョンをインストールして、その他のバージョンを上書きすることが一般的である。その結果、同じシステム上で実行されている別のアプリケーションが動作できなくなる可能性がある。

【００２５】
Ｗｉｎｄｏｗｓ（登録商標）　９８とＷｉｎｄｏｗｓ（登録商標）　２０００においてＭｉｃｒｏｓｏｆｔが開発したＷｉｎｄｏｗｓ（登録商標）　Ｐｒｏｔｅｃｔｅｄ　Ｆｉｌｅ　Ｓｙｓｔｅｍ（ＷＰＦＳ）により、システム管理者はアプリケーションのベースディレクトリにＸＸＸＸＬＯＣＡＬ（ＸＸＸＸは拡張子のない実行可能ファイル名）と呼ばれるファイルを作成できる。これにより、Ｗｉｎｄｏｗｓ（登録商標）　ＬｏａｄｅｒがＬｏａｄＬｉｂｒａｒｙの実行中にパス参照を解決する方法を変更する。ただし、問題を完全に解決するにはこれでは不十分である。不十分である第１の理由は、ＸＸＸＸファイルのセットアップの成否がシステム管理者の知識に左右されるからである。システム管理者の知識は、人によって大きな差がある。第２の理由は、コンポーネントのバージョンを元に戻してからローカルディレクトリにインストールし、次に”．ＬＯＣＡＬ”ファイルを作成しなければならないからである。これは、ＷＩＮＤＯＷＳ（登録商標）￥ＳＹＳＴＥＭに配置された最も基本的なコンポーネントを除いては、簡単なプロセスではない。さらに、この解決法では、必要とされる機能すべてに対処できるわけではない。ＬｏａｄＬｉｂｒａｒｙの実行中、Ｗｉｎｄｏｗｓ（登録商標）は、コンポーネントが明示的または非明示的なＬｏａｄＬｉｂｒａｒｙのどちらの結果として解決されたか、また、有名な（または周知の）ＤＬＬであることを示すＲｅｇｉｓｔｒｙ　Ｋｅｙが存在するかどうかに応じて、異なるパス解決セマンティクスを使用する。このケースでは、ＬｏａｄＬｉｂｒａｒｙ呼び出しは常にＷＩＮＤＯＷＳ（登録商標）￥ＳＹＳＴＥＭディレクトリに解決される。

【００２６】
ＤＬＬおよびその他の共有コンポーネントは、実行されているアプリケーションが参照していない場合はコンポーネントが変更されないことを保証するために、参照カウントセマンティクスも保持する。実際には、オペレーティングシステムベンダーのアプリケーションとオペレーティングシステム自体のみ、この規約に従うことができる。

【００２７】
原則として、常に共有オブジェクトが正しいコンポーネントに帰着するよう解決されることが望ましい。この機能を実現するには、アプリケーションがともに動作できるコンポーネントのバージョンまたはバージョンの範囲を知る必要がある。次に、アプリケーションの実行時に、本発明はコンポーネントが正しく解決されることを保証することが望ましい。本発明では、機能能力があれば、必要な場合はＷＰＦＳまたはその他のオペレーティングシステムを自動的に使用することも許容される。この場合、必要なコンポーネントを検出してローカルファイルシステムに配置する必要がある。この処理は、インストールの単なる監視よりも複雑である。なぜならば、必要なコンポーネントが既にインストールされている場合、インストールプログラムはコンポーネントをインストールしないことが多いからである。

【００２８】
名前付きオブジェクトが正しくロードされることも保証する方法を識別する必要性がある。Ｗｉｎｄｏｗｓ（登録商標）プラットフォームでは、ＭＳＶＣＲＴ．ＤＬＬがこの問題の大きな原因である。このオブジェクトの複数のバージョンが維持されている場合、上述のＲｅｇｉｓｔｒｙキーを動的に変更して、ＬｏａｄＬｉｂｒａｒｙ関数で正しいコンポーネントバージョンを解決することができる。正しいコンポーネントのロードを保証するもう１つの妥当な方法は、有効な検索パスを使用するようにプロセス環境を動的に編集することである。この検索パスによって、システムワイドコンポーネントの前にローカルコンポーネントが必ず解決される。正しい共有オブジェクトを解決する別の方法は、シンボリックリンクを使用することである。共有コンポーネントに対してシンボリックリンクを作成すれば、実行時にコンピュータのファイルシステムによって共有コンポーネントを必要なコンポーネントに帰着するよう解決することができる。最後の方法として、共有オブジェクトのファイルからの情報に対する実際のオープン／読み取り／クローズ要求を本発明によって中断し、ローカルシステム上または本発明のサブシステム内に存在する可能性

があるファイルの正しいバージョンに向けて動的に応答することができる。

【００２９】

いくつかの特別な形態が存在する。Ｗｉｎｄｏｗｓ（登録商標）プラットフォームでは、実行中の複数またはすべてのプロセス間でグローバルに共有するように、ＯＬＥ、ＯＤＢＣ、ＭＤＡＣ、およびその他のベンダー固有の多数のコンポーネントが書き込まれている。ＯＬＥの場合は、個別プロセス間でデータおよびメモリー空間でさえも共有される。これらのコンポーネントの多くと同様に、ＯＬＥはＯＬＥ自身の複数のコピーが一度に実行されることを防止する。またＯＬＥには、特定のアプリケーションについて特定バージョンのロードを必要とするバグと機能が多く存在する。本発明では、アプリケーションが必要とされるＯＬＥのバージョンをどれでもロードすることができ、同じバージョンのＯＬＥを使用する別のコンポーネントとセマンティクスを共有することも可能である。

【００３０】

一般に、明確に構成されていない場合は、競合防止のために共有オブジェクトをプライベートにロードする必要がある。コンポーネントをプライベートにロードするための方法は、コンポーネントの完全なアンロードや別のソフトウェアアプリケーション（オペレーティングシステムガードによってアクティブに管理されているかどうかにかかわらず）に対する正しいコンポーネントのロードを妨げてはならない。さらに、システムがクラッシュした場合、基本オペレーティングシステムが上書きまたは変更されていないクリーンな状態に回復する必要がある。

【００３１】

ファイル

多くのアプリケーションは、設定エントリまたはその他のアプリケーションデータを格納するためにアプリケーション内でデータファイルを使用する。本発明は、上記の仮想レジストリに酷似した仮想ファイルシステムを提供する。アプリケーションが起動する前に、本発明はファイルシステム変更内容のリストをロードすることができる。変更内容には、ファイルを隠すこと、仮想環境にファイルを追加すること、仮想環境内の別のファイルへファイルをリダイレクトすることなどが含まれる。アプリケーションがファイルにアクセスしたりファイルを変更した場合は必ずオペレーティングシステムガードが、そのファイルをリダイレクトする必要があるかどうかをチェックし、リダイレクトする必要がある場合、好ましい実施形態では、その要求をオペレーティングシステムガード構成で指定された場所にリダイレクトする。

【００３２】

ユーザーのローカルドライブに書き込むためにアプリケーションが新規ファイルの作成または既存ファイルのオープンを試みた場合、オペレーティングシステムガードは、リダイレクトされた場所でそのファイルが実際に作成または変更されることを保証しなければならない。後でアプリケーションが再ロードされる場合、このファイルマッピングをオペレーティングシステムガードの仮想環境に再ロードする必要がある。ユーザーのローカルドライブ上に常駐する既存ファイルの変更要求があった場合、オペレーティングシステムガードはそのファイルをリダイレクションポイントにコピーしてから、要求を継続しなければならない。ファイルパスの安全なマッピングを保証するために、リダイレクトされたファイルは元のファイルと同じ名前であってはならない。好ましい実施形態では、システムのセキュリティとアプリケーションの互換性を最大限確保するために、ＩＮＩファイルをこのように扱う。

【００３３】

本発明は、ネットワークを介して配布されるアプリケーションに対して特に有用である。このような実現化では、ソフトウェアアプリケーションが複数種類のデータから構成されることを理解することが重要であり、ソフトウェアアプリケーションで使用されるファイルの大部分を個別の論理ドライブに格納することが最も望ましい。ファイルベースとレジストリベースの両構成は、ユーザー固有であってもシステムワイドであってもよい。使用されるアプリケーション配布システムは、各ファイルがこれらの種類のいずれであるかを

マークする必要がある。この情報は、オペレーティングシステムガードシステムが適切に動作するために役立つ。

【００３４】

デバイスドライバ

多くのアプリケーションは、ハードウェアサポートまたはオペレーティングシステムとの低レベルの直接的相互作用など一部の機能を実現化するために、デバイスドライバまたはその他のオペレーティングシステムレベルソフトウェアを使用する。本発明では、オペレーティングシステムガードが、これらのコンポーネントをアプリケーションの仮想環境に対して動的に、そして、なるべくプライベートに追加および削除する機能を提供する。

【００３５】

多くのデバイスドライバは、動的にロード可能であるように構築される。可能であればすべてのデバイスドライバを動的にロードすることが、好ましい実施形態である。起動時にデバイスドライバを静的にロードする必要がある場合、アプリケーションを実行する前にユーザーはこの知識を得ていなければならない。システムを再起動した場合、アプリケーションを前回停止したところから継続することが望ましい。しかし、デバイスドライバのほとんどは動的にアンロードすることができない。ドライバを動的にアンロードしないことが望ましいが、これが実現できない場合は、次回の再起動時に削除するようにドライバがマークされる。ユーザーにこのことを通知することが望ましい。次回の再起動の前のアプリケーション実行が２度目である場合、システムはドライバが存在することを認識し、２度目のインストールを試みずに、終了を待ってから次回の再起動時に削除可能なコンポーネントを認識することが望ましい。

【００３６】

本発明を正しく機能させるには、各デバイスドライバクラスについて基本的な類似点と相違点を特徴づけることが大切である。常駐するシステムハードウェア用のデバイスドライバをロード・アンロードすることは望ましくない。プログラミングの容易さという点ではこれは好ましい実施形態ではないが、本発明の範囲を逸脱するものではなく、特別な理由で必要とされることがある。たとえば、本発明を使用して配布・実行されるアプリケーションの使用許諾を制限する場合である。

【００３７】

Ｍｉｃｒｏｓｏｆｔ以外のプラットフォームでは、デバイスドライバの扱いが異なることが多い。Ｍａｃｉｎｔｏｓｈシステムは静的ドライバと動的ドライバの双方をサポートするが、両ドライバはすべて同じ方法でインストール・削除される。Ｍａｃｉｎｔｏｓｈシステムフォルダにリンクすることにより、必要なサポートが提供される。ＵＮＩＸ（登録商標）システムではほとんどの場合、デバイスドライバを使用するために、実行中のＵＮＩＸ（登録商標）カーネルを変更した後で再起動しなければならない。このプロセスは非常に複雑な場合がある。好ましい実施形態では、このプロセスが自動化される（アプリケーション完了後のカーネルリセットなど）。このプロセスの一般的条件は、Ｗｉｎｄｏｗｓ（登録商標）アプリケーションについて説明した条件（実際の編集処理段階）と同じであり、このようなオペレーティングシステムに熟知したユーザーは再起動を実行することができる。

【００３８】

最後に、システム障害が発生した場合にドライバを回復・削除できることが望ましい。したがって、システムの完全性を保持するために必要なデータまたはプロセスはすべて本発明の好ましい実施形態である。さらに、いずれの種類のデバイスドライバも本発明によって都合良くまたは効果的に提供されるとは限らず、このことは、特に常設のハードウェアの付加デバイスに関連するものに成り立つこともまた当業者に了解されることである。

【００３９】

その他の項目

本発明では、代替オペレーティングシステム上ではそれらの動作または存在形態が異なるものとなる発明のいくつかのコンポーネントのあることが認められる。これらのコンポー

ネントには、フォント、プロセス、環境変数などが含まれる。

【００４０】

一部のアプリケーションでは、正しく実行するためにフォントをインストールしなければならない。必要なフォントはいずれも、オペレーティングシステムガードの構成ファイルで指定される。オペレーティングシステムガードはアプリケーションの実行に先立ってこれらのフォントを有効にし、必要であれば、実行後にフォントを削除する。ほとんどのシステムには、フォントを登録するプロセスまたはフォントの存在をシステムに通知するプロセスの他に、フォントを格納する共有領域があり、オペレーティングシステムガードはこれらの利用可能な方法を活用する。

【００４１】

Ｗｉｎｄｏｗｓ（登録商標）では、フォントは￥ＷＩＮＤＯＷＳ（登録商標）￥ＦＯＮＴＳディレクトリにコピーされる。ただし、これによって、実行中のプログラミングでそのフォントが利用できることが保証されるわけではない。好ましい実施形態では、プログラムがＷｉｎｄｏｗｓ（登録商標）　ＡＰＩを使用してフォントにアクセスする場合、ＣｒｅａｔｅＳｃａｌａｂｌｅＦｏｎｔＲｅｓｏｕｒｃｅ／ＡｄｄＦｏｎｔＲｅｓｏｕｒｃｅなどのＷｉｎ３２　ＡＰＩ呼び出しでフォントを登録する必要がある。これにより、フォントがシステムフォントテーブルに挿入される。挿入が完了すると、オペレーティングシステムガードは別の適切なＡＰＩ呼び出し（ＲｅｍｏｖｅＦｏｎｔＲｅｓｏｕｒｃｅなど）でフォントを削除し、システムからファイルを削除することができる。別の実施形態として、オペレーティングシステムガードは仮想レジストリ方法で説明したようにＡＰＩ関数をフックすることができる。さらに、オペレーティングシステムガードはファイルサブシステムを使用して、実際のフォントファイルが実行されているシステムに配置されることを防止することができる。

【００４２】

Ｍａｃｉｎｔｏｓｈでは、このプロセスは酷似しており、Ｍａｃｉｎｔｏｓｈシステムフォルダ内のファイルと登録アクティブ化に基づいている。しかしＵＮＩＸ（登録商標）では、このプロセスはアプリケーションに依存する。最も一般的には、適切な場所で解決された通常ファイルとしてフォントリソースがシステムに追加されるため、フォントリソースに名前でアクセスすることができる。多くのＭｏｔｉｆシステムでは、フォント記述をフォントリソースファイルに配置する必要がある。これにより、フォントの解決が可能になる。ＭｏｔｉｆまたはＸアプリケーションは、解決サブシステムまたはた直接呼び出しのいずれかでフォントを呼び出すことができる。最近では、ＭｏｔｉｆおよびＣＤＥベースシステムの多くで、Ａｄｏｂｅスケーラブルポストスクリプトフォントが利用されている。このフォントは、Ａｄｏｂｅタイプ管理システムで管理する必要がある。ただし例外があり、上述のように、Ｗｉｎｄｏｗｓ（登録商標）またはその他のオペレーティングシステムの既定フォント管理システムに代わるシステムが存在する。Ａｄｏｂｅ　ＴｙｐｅＭａｎａｇｅｒは、他のサードパーティのタイプ管理システムと同様に、このプロセスに代替インタフェースを提供する。ほとんどの場合、このインタフェースをサポートするか無視するかを決めることが望ましい。オペレーティングシステムガードの目的は、これらのシステムすべてに対応する汎用インタフェースを提供することではなく、オペレーティングシステム固有のサブシステムに対応するインタフェースのみ提供することである。

【００４３】

環境変数の設定を必要とするアプリケーションが多い。このことはＵＮＩＸ（登録商標）システムに最もよく当てはまるが、当初ＵＮＩＸ（登録商標）上で書き込まれＷｉｎｄｏｗｓ（登録商標）オペレーティングシステムに移植されたソフトウェアでも環境変数が頻繁に使用される。Ｗｉｎｄｏｗｓ（登録商標）オペレーティングシステム上のアプリケーションはＤＯＳ　ＰＡＴＨ環境変数に強く依存し、アプリケーション固有のエントリを設定することが多い。Ｗｉｎｄｏｗｓ（登録商標）　９ｘ／Ｍｅ環境では、基本的にはＤＯＳサブシステムであるため適用可能である多くの環境設定が存在する。アプリケーションが特定の変数を必要とする場合、または既存の環境変数に値を設定する必要がある場合、

必要な環境変数はオペレーティングシステムガードの構成ファイルで指定される。オペレーティングシステムガードは、起動時にアプリケーションのメインプロセスに対してこれらの変数を設定する。アプリケーションは通常、動作時に環境変数を変更しないため、仮想環境はこれらの呼び出しを捕捉せず、レジストリと構成サブシステムが提供する機能一式も提供しない。

【００４４】

回復

これまで示してきたいくつかのケースでは、オペレーティングシステムを実際に変更しなければならない。デバイスドライバとフォントが変更されることが多い。さらに、次回のアプリケーション実行時にも存続し利用可能でなければならない仮想環境を変更することもできる。オペレーティングシステムガードはシステムに対する変更を回復し、できるだけ早くシステムから変更を除去できなければならない。別の方法では、アプリケーション実行時にシステムがクラッシュした場合、オペレーティングシステムガードは、再起動時などにシステムに対する変更を除去するために十分な情報を追跡し、仮想環境に対する変更を追跡することが望ましい。好ましい実施形態では、この追跡はトランザクションログとして実現化される。ただし別の実施形態では、変更を取り消すことができるようにシステムの起動時に読み取ることができる他の類似コンポーネントとして実現化することもできる。

【００４５】

仮想化の制御

本発明の重要な態様は、オペレーティングシステムガードが実行できる仮想化の様々な局面の制御に関する。好ましい実施形態では、ソフトウェアシステムの正しい状態を確実に制御できる計装プログラムが存在する。また、システムによって仮想化される項目を管理者とエンドユーザーが視認および変更できるようにする方法も含まれる。

【００４６】

自動プログラムでは、制御の状態を評価するために制御対象のアプリケーションが監視される。自動プログラムは、アプリケーションのインストールプロセス時、アプリケーションの実行時、またはその両方でこの監視タスクを実行できる。好ましい実施形態では、オペレーティングシステムガードがラッパーアプリケーションに組み込まれる。インストール後、またはソフトウェアを１回以上使用した後、ラッパーアプリケーションはソフトウェアの活動すべての詳細リストについてオペレーティングシステムガードに問い合わせる。この活動リストを基に、ラッパーアプリケーションは、以降の使用時にオペレーティングシステムガードをロードし作動させるために必要な構成ファイルを作成する。

【００４７】

好ましい実施形態では、オペレーティングシステムをインストールプロセスの一部として使用する場合、オペレーティングシステムガードは、インストールをその環境にのみ入れることを許可する仮想レイヤとして機能する。インストール後、後の再ロードに備えてすべてのファイルや設定などをダンプすることができる。この方法では、インストールによって元のシステムが変更されることはなく、必要な構成ファイルが自動的に作成される。アプリケーションの使用中にオペレーティングシステムガードを使用すると、環境に対する差分変更の記録、または構成ファイルの再作成が可能になる。

【００４８】

オペレーティングシステムガードは、後処理のために情報をラッパーアプリケーションに渡す。好ましい実施形態では、システムが作成できる自動エントリに加え、オペレーティングシステム固有、アプリケーション固有、またはドメイン固有の知識でラッパーアプリケーションがプログラムされる。この知識は、構成項目またはその他のエントリの周知の使用状況を反映するようにプロセスの出力を変更するために使用される。好ましい実施形態では、コーディングを変更するために、ルールベースシステムを利用して、監視された動作と周知のシナリオを比較する。

【００４９】

ラッパーアプリケーションは、プロセスの構成出力に対するビューアおよび／またはエディタとしても使用される。好ましい実施形態では、このエディタを使用してシステム管理者は、構成に対する項目または項目グループの追加、編集、または削除ができる。エディタによって構成を観察する際、管理者は構成の複製を作成し、アプリケーションレベルの変更またはユーザー独自の変更を行うために必要に応じて特定の項目を変更することもできる。

【００５０】

図１では、本発明の一実施形態の機能を示している。この実施形態では、アプリケーション／ユーザーデータ（６０）が２セットある。オペレーティングシステムガード（１００）は、アプリケーション（５０）の２つのインスタンスが互いに干渉することを防止する。また上述のように、オペレーティングシステムガード（１００）は抽象化レイヤとして機能し、クライアントコンピュータのアプリケーションソフトウェア（５０）と実際のオペレーティングシステム（１０）との間でコマンドと通信を収集する。矢印で示したように、一部のコマンドはオペレーティングシステムガードとソフトウェアアプリケーションとの間にある。このことが、一般的なインストールと違う点である。一般的なインストールでは、これらのコマンドはオペレーティングシステム自身によって実行されるため、オペレータの意図とは必ずしも一致しない変更がクライアントコンピュータに生じる。一方、その他のコマンドはオペレーティングシステムガードを経由してオペレーティングシステム自身に転送される。

【００５１】

好ましい実施形態と合わせて本発明を詳述してきたが、添付した請求項によって網羅される本発明の範囲から逸脱することなく形態と詳細を様々に変更できることは当業者にとって理解されるであろう。

【図面の簡単な説明】

【００５２】

【図１】本発明、オペレーティングシステム、およびソフトウェアアプリケーションの相対的関係を示すブロック略図である。

【図２】を示すブロック略図である。

【図３】を示すブロック略図である。

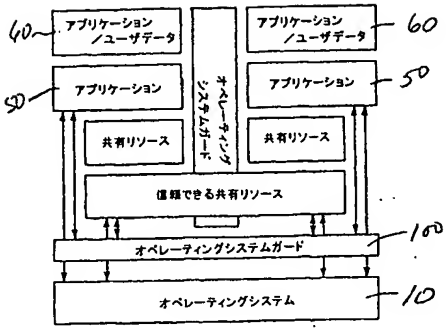
【図４】を示すブロック略図である。

【符号の説明】

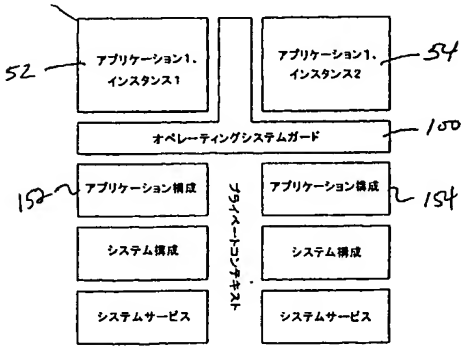
【００５３】

１０：オペレーティングシステム
５０：アプリケーション
５２：アプリケーション１、インスタンス１
５４：アプリケーション１、インスタンス２
６０：アプリケーション／ユーザーデータ
１００：オペレーティングシステムガード
１０２：コア
１０４：構成マネージャ
１０６：ファイルマネージャ
１０８：共有オブジェクトマネージャ
１１０：デバイスマネージャ
１１２：フォントマネージャ
１１４：プロセス環境マネージャ
１１６：ローダー（サブシステム）
１１８：リカバリマネージャ
１２０：プロセスマネージャ
１５２：構成
１５４：構成

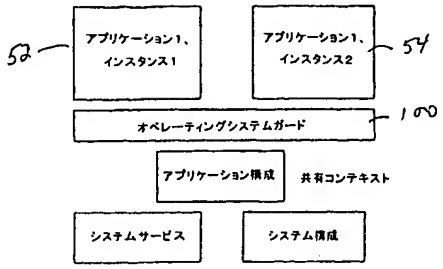
【図１】


アプリケーション／ユーザデータ
アプリケーション／ユーザデータ
60
60
アプリケーション
アプリケーション
50
50
オペレーティングシステムガード
共有リソース
共有リソース
信頼できる共有リソース
オペレーティングシステムガード
100
オペレーティングシステム
10

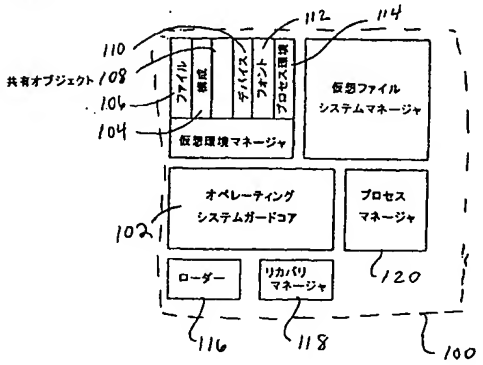
【図２】


アプリケーション1、インスタンス1
52
アプリケーション1、インスタンス2
54
オペレーティングシステムガード
100
アプリケーション構成
152
アプリケーション構成
154
プライベートコンテキスト
システム構成
システム構成
システムサービス
システムサービス

【図３】


アプリケーション1、インスタンス1
52
アプリケーション1、インスタンス2
54
オペレーティングシステムガード
100
アプリケーション構成
共有コンテキスト
システムサービス
システム構成

【図４】


110
112
114
共有オブジェクト
108
106
104
ファイル
領域
デバイス
フォント
プロセス環境
仮想ファイルシステムマネージャ
仮想環境マネージャ
オペレーティングシステムガードコア
102
プロセスマネージャ
120
ローダー
リカバリマネージャ
116
118
100

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 November 2002 (21.11.2002)

PCT

(10) International Publication Number
WO 02/093369 A1

(51) International Patent Classification⁷: G06F 9/445, 9/44, 15/163, 17/30

(21) International Application Number: PCT/US02/15378

(22) International Filing Date: 15 May 2002 (15.05.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/859,209 16 May 2001 (16.05.2001) US

(71) Applicant: SOFTRICITY, INC. [US/US]; 332 Congress Street, Boston, MA 02210 (US).

(72) Inventor: SCHAEFER, Stuart; One Gallison Avenue, Marblehead, MA 01945 (US).

(74) Agents: KEYACK, Albert, T. et al.; Schnader Harrison Segal & Lewis, LLP, 1600 Market Street, 36th Floor, Philadelphia, PA 19103 (US).

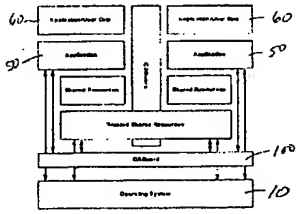
(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States (regional): European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— with international search report
— before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: OPERATING SYSTEM ABSTRACTION AND PROTECTION LAYER



(57) Abstract: The present invention provides a system for creating an application software environment without changing an operating system of a client computer, the system comprising an operating system abstraction and protection layer, wherein said abstraction and protection layer is interposed between a running software application and said operating system, whereby a virtual environment in which an application may run is provided and application level interactions are substantially removed. Preferably, any changes directly to the operating system are selectively made within the context of the running application and the abstraction and protection layer dynamically changes the virtual environment according to administrative settings. Additionally, in certain embodiments, the system continually monitors the use of shared system resources and acts as a service to apply and remove changes to system components. The present thin invention defines an "Operating System Guard." These components cover the protection semantics required by DLLs and other shared library code as well as system device drivers, fonts, registries and other configuration items, files, and environment variables.

WO 02/093369 A1

OPERATING SYSTEM ABSTRACTION AND PROTECTION LAYER

This application is a continuation-in-part of U.S. Patent Application No. 09/456,181, the entirety of which is incorporated herein by reference as if fully set forth.

The present invention relates to computer software, and more particularly to operating system software.

BACKGROUND OF THE INVENTION

In many environments, but particularly in environments where an application is delivered via a network, the most important feature is an ability to run applications on the fly, without a complex installation. Typically, in certain prior art systems, great pains were taken to modify a client system to appear as if a program was installed, or to actually install the software itself, and then back out these modifications to restore the original configuration. In doing this, multiple problems present themselves: conflicts between an application and the computer's current configuration, multiple instances of the same or different applications, complexity of the back out process requires an application to be put through a rigurous process to ensure all of its modifications can be accounted for, and the use of shared files and system components by multiple applications complicates back out and the installation process.

SUMMARY OF THE INVENTION

The present invention provides a system for creating an application software environment without changing an operating system of a client computer, the system comprising an operating system abstraction and protection layer, wherein said abstraction and protection layer is interposed between a running software application and said operating system, whereby a virtual environment in which an application may run is provided and application level interactions are substantially removed. Preferably, any changes directly to the operating system are selectively made within the context of the running application and the abstraction and protection layer dynamically changes the virtual environment according to administrative settings. Additionally, in certain embodiments, the system continually monitors the use of shared system resources and acts as a service to apply and remove changes to system components.

Thus, for example, in embodiments within Windows-based operating systems, and wherein all operations to the Windows Registry are through the Win32 API, the system preferably provides a means for hooking functions, whereby each time said functions are invoked another function or application intercepts the call, and the system most preferably hooks each appropriate API function to service a request whether made by an application run from a server or if made by an application against a configuration key being actively managed

In other preferred embodiments of the present invention, additional functionality is provided, such as those embodiments wherein the operating system abstraction and protection layer manages the integration of multiple instances of an application by recognizing how many instances of an application are running, and in such embodiments most preferably it also avoids making changes on startup and shutdown unless there is only one application instance running. In this embodiment it is also possible to support multi-user operating systems in which multiple instances of an application can be running on behalf of different users.

Thus, the operating system abstraction and protection layer presents an environment to an application that appears to be an installation environment without performing an installation, whereby a "pseudo installation" is created in which all of the settings are brought into a virtual environment at the time the application runs. Or in the case of an installed application, acts to dynamically modify the behavior of the application at run-time. Preferred embodiments provide a means for preventing information on the client computer from interfering or modifying the behavior of an application, and most preferably provide a means for dynamically changing the virtual environment according to administrative settings. As mentioned above, in certain embodiments it will be possible to have more than one instance of a single software application running on the same client computer, even if it was not originally authored to do so. In such embodiments, shared, controlled contexts are provided in which at least two of said instances of a single application share one or more virtual settings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram schematic showing the relative relationship of the present invention, an operating system and a software application;

FIG. 2 is a block diagram schematic showing

FIG. 3 is a block diagram schematic showing

FIG. 4 is a block diagram schematic showing

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to FIG. 1, there is illustrated a block diagram schematic showing the relative relationship of the present invention, an operating system and a software application. Preferred embodiments of the present invention provide an operating system abstraction and protection layer 100 denominated an "Operating System Guard." Internally, many operating systems 10 provide fault domains to protect applications 50 from affecting each other when run. However, shared system resources and many other operating system features allow this protection domain to be compromised. An operating system abstraction and protection layer 100 will provide an additional, programmatically controlled barrier between applications 50 to

remove most application level interactions. Disposed between the application 50 and operating system 10 the operating system abstraction and protection layer 100 selectively allows changes directly to the operating system 10, versus containing the change within the context of the running application. For one example, in Windows-based systems, all operations to the Windows Registry are typically done through the Win32 API. As explained below, system functions like QueryRegEx and GetProfileString can be hooked so that each time they are invoked, another function or application intercepts the call. The Operating System Guard 100 of the present invention will hook each appropriate API function to service the request, if made by an application being actively managed or if made by an application against a configuration item being actively managed. In this way, unless explicitly configured to do so, the present invention can create the application environment without making any actual changes to the end-user's system. Also, any modifications made at run-time by the application can be persisted or removed easily.

As used herein the term "Operating System Guard" defines a layer between a running application and the operating system of a target computer or client computer that provides a virtual environment in which an application may run. This virtual environment has several purposes. First, it prevents a running application from making changes to the client computer. If an application attempts to change underlying operating system settings of a client computer, such settings are protected and only "made" in the virtual environment. For example, if an application attempts to change the version of a shared object like MSVCRT.DLL, this change is localized to the application and the code resident on the client computer is left untouched.

Second, the invention presents an environment to a running application that appears to be an installation environment without performing an installation, and is thus a "pseudo installation" or "installation-like." All of the settings are brought into a virtual environment at the time the application being served runs, or just-in-time when the application needs the particular setting. For example, if a computer program such as Adobe Photoshop® expects to see a set of Windows Registry entries under HKEY_LOCAL_MACHINE\Software\Adobe and they are not there on the client

computer since Photoshop was never installed, a system made in accordance with this aspect of the present invention will "show" those registry entries to the Photoshop programming code exactly as if they were resident on the client computer.

Next, the invention prevents information that may exist on the client/users machine from interfering with or modifying the behavior of an application. For example, if the user has already existing registry entries under:

HKEY_LOCAL_MACHINE\Software\Adobe

for an older version of Photoshop, but now wishes to operate a newer version, these entries can be hidden from the new application to prevent conflicts.

Finally, the present invention unlocks application behavior that may not exist as the application is currently written. It does this through the ability to dynamically change the virtual environment according to administrative settings. For example, in a typical instance of an enterprise software application, a client application may expect to read a setting for the address of the database to which the user should connect from a setting in the registry. Because this registry key is often stored in HKEY_LOCAL_MACHINE, the setting is global for the entire client computer. A user can only connect to one database without reinstalling the client, or knowing how to modify this registry key, and doing so each time they wish to run the application. However, by implementing the present invention, two instances of the application may now run on the same client computer, each connecting to a different database.

CONTEXTS

In providing this functionality, each application is able to run in a private context within the system. To the application, it has its own private view of what the system looks like and its behavior. The present invention provides this by its inherent nature. Referring to FIG. 2, two separate applications 52,54, or two instances of the same application (50 illustrated in FIG. 1), can be provided private contexts in which they will appear to have separate or differing copies of system services, configuration and data. In the preferred embodiment, this is the default behavior of the system.

By extending this concept, the Operating System Guard 100 of the present invention can also provide shared, controlled contexts in which two or more applications 52,54 can share some or all of their virtual settings. This is important for application suites such as Microsoft Office, or for applications that perform differently in the presence of other applications. For example, many applications use Microsoft Word as an engine for doing Mail Merge or document creation functionality. The application must know about the installation or presence of Word and be able to tap into its functions. In the preferred embodiment, two instances of the same application will share a single context by default, while two separate applications will maintain private contexts. Referring to FIG. 3, the two applications 52,54 can run while the Operating System Guard 100 provides a shared view of the available system resources.

DESIGN

As illustrated in FIG. 4, the Operating System Guard is comprised of the following subsystems: core 102, configuration manager 104, file manager 106, shared object manager 108, device manager 110, font manager 112, process manager 120, process environment manager 114, loader 116, and recovery manager 118. With the exception of the core 102, the process manager 120, and the loader 116, all other subsystems are elements of the Virtualization System described in further detail below. The core 102 is primarily responsible for managing applications and their context as defined by the configuration files.

The process manager 120 provided by the Operating System Guard allows the core 102 to be informed of any process or thread event that may be of interest. It also provides an abstraction layer to the operating system-dependent implementations for managing a process space and handling thread processing. Processes may be grouped together into application bundles. An application bundle is a group of processes which all share their virtual resources with each other. For example, Microsoft Word and Microsoft Excel may want to share the virtual registry and virtual file system to be able to work together as an application suite. The process manager 120 calls these application bundles

"applications". The information about an application exists until the process manager 120 is told to release the application. If another process needs to be loaded into the application bundle, it may do so as long as the application has not been released.

The loader subsystem 116 of the present invention is used to allow virtual environments to be transferred into and out of the running system. Each of the Virtualization Subsystems is capable of serializing its configuration for the loader 116, and retrieving it through the reverse process. In addition, the loader 116 is capable of staged loading/unloading and combining the results of individual stages into one single environment description.

REGISTRY AND CONFIGURATION

Applications require varying amounts of configuration information to operate properly. Anywhere from zero to thousands of configuration records exist for which an application can read its configuration On Windows, there are two common places for configuration information, the Windows Registry and system level initialization files win.ini and system.ini. In addition, the \WINDOWS\SYSTEM directory is a common place for applications to write application specific configuration or initialization files. Applications will also use configuration or data files in their local application directories to store additional configuration information. Often this information is difficult to deal with, as it is in a proprietary format. On platforms other than Windows, there is no equivalent of the Registry, but common directories exist for configuration information. X Windows has an app-defaults directory. Macintosh has the System Folder, and other operating systems will have corresponding elements. It is important to note that on most UNIX systems, each individual application 52,54 will most often store its own configuration 152,154 locally, as seen in FIG. 2.

The present invention, in one embodiment, includes a virtual Windows Registry component, which will provide a full function registry to an application, but prevent modification to the underlying system registry. All keys that an application expects to access will be present, but may only exist in the virtual registry. In this way, the

Operating System Guard 100 of the present invention and the Windows Registry form a two-stage process for accessing the registry. If an application needs access to a key, it will query the Registry. The Operating System Guard will respond with the key and its value if it knows it. Otherwise, it will allow the request to pass through to the Windows Registry. If an attempt is made to modify the value, the Operating System Guard will allow the modification to occur to itself only. The next time the application accesses the key, it will be present in the Operating System Guard and the request will not flow through to the real Registry, leaving it untouched.

The keys that the Operating System Guard uses are specified in three separate sections. These Operating System Guard keys are specified as commands in these sections to modify an existing key, delete the presence of a key, or add a new key to the registry. In this way, the virtual registry can appear exactly as the system intends. This is important as the presence or absence of a key can be as important as the actual value of the key.

In the preferred embodiment, the Operating System Guard first loads a data file that contains basic registry entries for the application. Then a second data file is loaded that contains the user's preferences. Finally, the Operating System Guard can optionally load a set of keys that include policy items that the user is not allowed to override. The three files load on top of each other with duplicate items in each file overriding items in the file before it. The first time a user runs an application, the second data file will not exist because there will be no user-specific information, only application defaults. After each session, though, the Operating System Guard will save the user's changes, generating that second data file for use in future sessions.

Configuration files can be modified in two ways. First, the file can be edited directly by an application. In this scenario, the Operating System Guard File subsystem described below will address the modification made to the file. Second, in the preferred embodiment, an application can call the Windows API family of calls GetProfileString, WriteProfileString, or others to modify these files. In this case, the Operating System

Guard of the present invention performs exactly as described above intercepting these calls and servicing them from within.

SHARED OBJECTS

Many components used by operating systems and running applications are shared across several applications or instances. In general, this is a very good idea. It saves disk space, not requiring many copies of the same file. It also provides the ability for operating system vendors and third parties to create and distribute libraries of commonly used code. On the Windows platform, Dynamic Link Libraries, DLLs, are often shared within and across applications. On other platforms, the problem is the same. On the Macintosh, INITs and other system components are loaded for applications. These components can have many versions, of which only one is used at a time. On UNIX systems, dynamic shared objects, e.g., ".so" library files, are used by applications to speed load time, save disk space, and for other reasons. Many programs use the default "libc.so." However, this library file is typically a symbolic link to some version of itself such as libc.so.3. In practice, this feature has created havoc. These shared components have often gone through revision, with many versions of the same component available to be installed. Application authors have found their software to work with potentially only one or some of the versions of the shared component. Thus, in practice, applications typically install the version they desire, overwriting other present versions. This potentially causes defaults in other applications running on a system.

On Windows 98, Windows 2000, Microsoft has created the Windows Protected File System (WPFS) to allow system administrators to create a file called XXXX.LOCAL in the base directory of an application, where XXXX is the executable file name without the extension. This causes the Windows Loader to alter its method of resolving path references during LoadLibrary executions. This, however, is not sufficient to completely solve the problem. First, setting up the XXXX file is left to the knowledge of the system administrator, which varies widely. Second, a component version must undergo a rewind back to the original, then install this component in the local directory, and then create the ".LOCAL" file. This is not a straightforward process for any but the

most basic components placed in WINDOWS\SYSTEM. Also, this solution does not cover all of the needed functionality. During LoadLibrary, Windows uses different path resolution semantics depending on whether the component was resolved as a result of an explicit or implicit LoadLibrary, and also whether a Registry Key exists indicating that it is a named, or well-known, DLL. In this case, the LoadLibrary call will always resolve to the WINDOWS\SYSTEM directory.

DLLs and other shared components also retain reference count semantics to ensure that a component is not touched unless no running applications refer to it. In practice, only applications from the operating system vendor and the operating system itself have done a good job of obeying this protocol.

As a general rule, it is desired to have a shared object always resolve to the correct component. To provide this functionality it is required to understand the version of a component, or range of versions, that an application is able to function with. Then, when the application is to be run, the present invention should ensure that the component is resolved correctly. It is acceptable, in the present invention, to automate the use of WPFS or other operating system provided capability, if desired. In this case, it is necessary to detect needed components and place them in the local file system. This is more complex than just watching installation, as an installation program will often not install a component if the required one is already there.

It is desired to identify a method to ensure that named objects are also loaded correctly. On the Windows platform, MSVCRT.DLL is a significant culprit within this problem area. If multiple versions of this object are maintained, the aforementioned Registry key can be dynamically changed, allowing the LoadLibrary function to resolve the correct component version. Another reasonable method of ensuring correct component loading is the dynamic editing of a process environment to use a valid search path. This search path will ensure that a local component is resolved before a system wide component. Another possible method for resolution of the correct shared object is through the use of symbolic links. A symbolic link can be made for a shared component,

which is resolved at run-time by the computer's file system to the needed component. Finally, the actual open/read/close requests for information from a shared object's file can be intercepted by the present invention and responded to dynamically for the correct version of the file which may exist on the local system or within the invention's subsystems.

Several special forms exist. On the Windows platform, OLE, ODBC, MDAC, ... as well as a number of other vendor specific components, are written to be shared globally among several or all running processes. In the case of OLE, going as far as sharing data and memory space between separate processes. OLE prevents more than one copy of itself running at a time, as do many of these components. OLE also has many bugs and features requiring a specific version to be loaded for a specific application. In the present invention, an application is able to load whatever version of OLE is required, still enabling the shared semantics with other components using the same version of OLE.

In general, unless specifically configured as such, shared objects should be loaded privately to ensure conflict prevention. Nothing about the method used to allow a component to be loaded privately should prevent it from being unloaded cleanly or correctly loading for another software application, whether being actively managed by the Operating System Guard or not. In addition, if the system crashes it is required to recover from this crash to a clean state, not having overwritten or modified the underlying operating system.

FILES

Many applications use data files within the application to store configuration entries or other application data. The present invention provides a virtual file system much like the virtual registry described above. Before the application starts, the present invention can load a list of file system changes, including files to hide and files to add to the virtual environment or files to redirect to another within the virtual environment. Whenever the application accesses or modifies any files, the Operating System Guard

checks if the file must be redirected, and if so, in the preferred embodiment redirects the request to a location specified in the Operating System Guard configuration.

If an application tries to create a new file or open an existing file for writing on a user's local drive, the Operating System Guard must ensure that the file is actually created or modified in the redirected location. If the application is reloaded at a later time, this file mapping must be reloaded into the Operating System Guard virtual environment. When the request is to modify an existing file, which resides on a user's local drive, the Operating System Guard must copy the file in question to the redirection point before continuing with the request. The redirected files may not be of the same name as the original file to ensure safe mapping of file paths. In the preferred embodiment, INI files are handled in this way to offer maximum system security while allowing maximum application compatibility.

The present invention is particularly useful for applications delivered over a network. In such implementations it is important to understand that software applications are made of several kinds of data, where the bulk of the files a software application uses are most preferably mounted on a separate logical drive. Configuration, including both file based and registry based, can be user specific and system wide. The application delivery system used should mark each file for which of these types any file is. This information provides hints to the Operating System Guard system to act on appropriately.

DEVICE DRIVERS

Many applications use device drivers or other operating system level software to implement some of its functions such as hardware support or low level interactions directly with the operating system. In the present invention, the Operating System Guard will provide the capability of dynamically, and as possible privately, adding and removing these components to an application's virtual environment.

Many device drivers are built to be dynamically loadable. If at all possible, it is the preferred embodiment to load all device drivers dynamically. If a device driver

requires static load at boot time, the user must be presented with this knowledge before running the application. Once the system has rebooted, the application should continue from where it left off. However, a large percentage of device drivers are not dynamically unloadable. Although it is preferred to dynamically unload the driver, if this cannot be accomplished the driver will be marked for removal on the next reboot, and the user should be made aware of this. If the application is run a second time before the next reboot, the system should remain aware of the presence of the driver and not attempt a second installation, waiting for termination to remark the component removable at next reboot.

It is important to characterize the base similarities and differences, as they exist for each device driver class, to ensure the present invention can correctly function. It is not truly desired to load and unload device drivers for system hardware that is constantly present. It should be understood that although this is not a preferred embodiment in terms of programming ease, it is within the scope of the present invention and may be required for specific reasons, such as the restriction in licensing agreements for applications that are delivered and run using the present invention.

On non-Microsoft platforms, device drivers are typically handled very differently. Macintosh systems support both static and dynamic drivers, but they are all installed and removed through the same method. Linking with the Macintosh system folder will provide the necessary support. For UNIX systems, device drivers most typically require a modification to the running UNIX kernel, followed by a reboot. This process can be very complex. In the preferred embodiment, this process is automated; including resetting the kernel once the application is complete. The general parameters of the process are the same as that described above for Windows applications, the actual process steps of compilation and persons familiar with such operating systems can carry out reboot.

Finally, those of skill in the art will understand that it is desirable to be able to recover and remove drivers across system failures. Whatever data or processes necessary to retain system integrity are therefore a preferred embodiment of the present invention. Those of skill in the art will also appreciate that all types of device drivers might not be conveniently or efficiently provided via the present invention, most particularly those associated with permanent hardware attached devices.

OTHER ITEMS

In the present invention, it is recognized that there are several components of the invention, the behavior or presence of which is different on alternate operating systems. These components include fonts, processes, environment variables, and others.

Some applications require fonts to be installed in order to perform correctly. Any fonts required will be specified in the Operating System Guard's configuration file. The Operating System Guard will enable these fonts prior to running the application and if necessary remove them afterwards. Most systems have a common area for storage of fonts in addition to a process for registering them or making the system aware of their presence, the Operating System Guard will utilize these available methods.

On Windows, a font is copied to the \WINDOWS\FONTS directory. This however does not guarantee that the font is available to the running program. In the preferred embodiment, if the program uses the Windows API to access fonts, the font will need to be registered with a Win32 API call such as CreateScalableFontResource/. AddFontResource. This will insert the font into the system font table. Once complete, the Operating System Guard can remove the font with another appropriate API call like RemoveFontResource, then remove the file from the system. As an alternate embodiment, the Operating System Guard could hook the API functions as described in the virtual registry method. In addition, the Operating System Guard can use its File subsystem to avoid placing the actual font file in the running system.

On Macintosh, the process is extremely similar and based on files in the Macintosh system folder and registration activation. On UNIX, however, the process is

dependent upon the application. Most typically, font resources are added to the system as regular files resolved in the proper location, so they can be accessed by name. With many Motif systems, a font description needs to be placed into a font resource file, which will allow the font to be resolved. The Motif or X application can invoke the font either through the resolution subsystem or by a direct call. Recently, many Motif and CDE based systems utilize Adobe scalable postscript fonts. These fonts need to managed through the Adobe type management system. There are exceptions, however, and as stated above, there are alternates to the Windows or other operating system default font management systems. The Adobe Type Manager provides some alternate interfaces for this process, as do other third party type management systems. In most cases it should be decided whether to support the interface or ignore it. The purpose of Operating System Guard is not to provide a universal layer for all these systems, only to do so for the operating system's own subsystem

Many applications require environment variables to be set. This is most common on UNIX systems, but is also heavily used by software, which was originally written on UNIX and ported to the Windows operating systems. Applications on the Windows operating systems heavily rely on the DOS PATH environment variable and often set their own application specific entries. On the Windows 9x/Me environments, there are many environment settings, which are applicable as at its core is the DOS subsystem. If an application requires the presence of specific variables, or values to be set in existing environment variables, the required environment variables will be specified in the Operating System Guard's configuration file. The Operating System Guard will set these variables for the application's main process when it is launched. As applications do not typically change environment settings as they operate, the virtual environment will not trap these calls, nor will it provide the full complement of functionality that the registry and configuration subsystem does.

RECOVERY

In some cases shown in the previous sections, actual modifications must be made to the operating system. This is frequent with device drivers and fonts. In addition,

changes can be made to the virtual environment that need to be persisted and available the next time an application is run. It is required that the Operating System Guard system be able to recover from changes to the system, removing the change from the system at its earliest possible opportunity. Alternately, if the system crashes during an application's execution, the Operating System Guard should track enough information to remove any change to the system if it is rebooted or otherwise, and should track the changes made to the virtual environment. In the preferred embodiment, this is implemented as a transaction log, but can in other embodiments be done as some other similar component, which can be read on system startup so that changes can be backed out.

CONTROLLING VIRTUALIZATION

An important aspect of the invention relates to control of the many facets of virtualization which the Operating System Guard is capable of. In the preferred embodiment there exists an instrumentation program able to ascertain the correct aspects of a software system to control. Also included is a method to allow administrators and end users to view and modify those items to be virtualized by the system.

In the automated program, the application to be controlled is observed in order to gauge the aspects of control. The automated program is capable of performing this task during the installation process of the application, during run-time of the application, or a combination of both. In the preferred embodiment, the Operating System Guard is embedded in a wrapper application. Post installation, or after one or many uses of the software, the wrapper application will query the Operating System Guard for a detailed list of all of its actions. From this list of actions, the wrapper application will create the configuration files required to load and operate the Operating System Guard on subsequent uses.

If used as part of the installation process, the Operating System Guard, in the preferred embodiment, will act as a virtual layer allowing the installation to be entered into its environment only. After the installation, all of the files, settings, et. al. can be

dumped for reload later. In this way, the installation will leave the original system intact and will have automatically created the necessary configuration files. When used during use of the application, the Operating System Guard is able to record either differential modifications to the environment, or recodify the configuration files.

The Operating System Guard will pass its information to the wrapper application for post-processing. In the preferred embodiment, in addition to the automatic entries that the system can create, the wrapper application is programmed with operating system specific and application or domain specific knowledge. This knowledge is used to alter the output of the process to reflect known uses of configuration items or other entries. In the preferred embodiment, a rules-based system is employed to compare observed behaviors with known scenarios in order to effect changes to the coding.

The wrapper application is also used as a viewer and/or editor for the configuration output of the process. This editor, in the preferred embodiment, enables a system administrator to add, edit, or delete items or groups of items from the configuration. In observing the configuration through the editor, the administrator can also make replicas of the configuration, changing specific items as needed to effect application level or user custom changes

Referring now to FIG. 1, an embodiment of the present invention is illustrated functionally. In this embodiment, two sets of application/user data 60 are illustrated. The Operating System Guard 100 keeps the two instances of the application 50 from interfering with one another. In addition, as explained above, the operating system guard 100 serves as an abstraction layer and as such collects commands and communications between the application software 50 and the actual operating system 10 of the client computer. As illustrated graphically by the arrows, certain commands are between the Operating System Guard and the software application, this is in distinction to typical installations where these commands would instead be acted upon by the operating system itself, resulting in changes to the client computer that might not necessarily be what the

operator intended. On the other hand, other commands pass through the Operating System Guard and are then transferred to the Operating System itself.

While this invention has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims.

What is claimed is:

1. A system for creating an application software environment without changing an operating system of a client computer, the system comprising an operating system abstraction and protection layer, wherein said abstraction and protection layer is interposed between a running software application and said operating system, whereby a virtual environment in which an application may run is provided and application level interactions are substantially removed.

2. The system of claim 1, wherein changes directly to the operating system are selectively made within the context of the running application.

3. The system of claim 2, wherein the abstraction and protection layer dynamically changes the virtual environment according to administrative settings.

4. The system of claim 1, wherein the system continually monitors the use of shared system resources and acts as a service to apply and remove changes to system components.

5. The system of claim 1, wherein the operating system is a Windows-based operating system, and wherein all operations to the Windows Registry and .ini files are through the Win32 API, the system further comprising a means for hooking functions, whereby each time said functions are invoked another function or application intercepts the call.

6. The system of claim 5, wherein the system hooks each appropriate API function to service a request whether made by an application run from a server or if made by an application against a configuration key being actively managed

7. The system of claim 1, wherein said operating system abstraction and protection layer manages the integration of multiple instances of an application by recognizing how many instances of an application are running.

8. The system of claim 7, wherein said operating system abstraction and protection layer avoids making changes on startup and shutdown unless there is only one application instance running.

9. The system of claim 1, wherein said operating system abstraction and protection layer presents an environment to an application that appears to be an installation environment without performing an installation, whereby a "pseudo installation" is created in which all of the settings are brought into a virtual environment at the time the application runs.

10. The system of claim 9, further comprising a means for preventing information on the client computer from interfering or modifying the behavior of an application.

11. The system of claim 9, further comprising a means for dynamically changing the virtual environment according to administrative settings.

12. The system of claim 9, wherein more than one instance of a single software application runs on the same client computer, and wherein each of said more than one instance connects to a different database.

13. The system of claim 12, wherein shared, controlled contexts are provided in which at least two of said instances of a single application share one or more virtual settings.

14. The system of claim 1, further comprising a device driver monitor that receives instructions at the time of installation for a particular application.

15. The system of claim 1, further comprising a virtual Windows Registry component to provide a full function registry to an application, but prevent modification to the underlying system registry.

16. The system of claim 1, wherein the operating system abstraction and protection layer responds with a key and its value if said key and value are stored within the operating system abstraction and protection layer, if not stored, the operating system abstraction and protection layer allows the request to pass through to the Windows Registry.

17. The system of claim 16, wherein if an attempt is made to modify the value of said key, the operating system abstraction and protection layer allows the modification to occur to itself only.

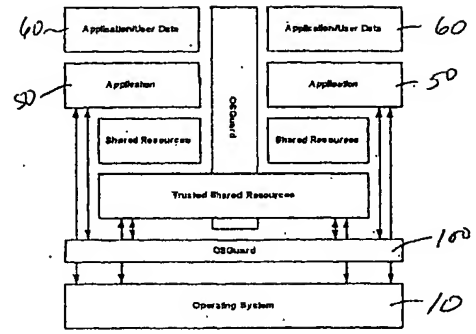

60
Application/User Data
Application/User Data
60
50
Application
Application
50
OSGuard
Shared Resources
Shared Resources
Trusted Shared Resources
100
OSGuard
10
Operating System

FIG. 1

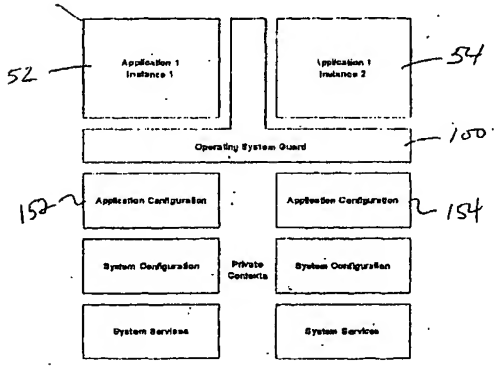

52
Application 1
Instance 1
Application 1
Instance 2
54
100
Operating System Guard
152
Application Configuration
Application Configuration
154
System Configuration
Private
Contexts
System Configuration
System Services
System Services

FIG. 2

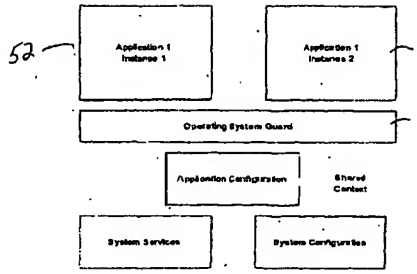

52
Application 1 Instance 1
Application 1 Instance 2
54
Operating System Guard
100
Application Configuration
Shared Context
System Services
System Configuration

FIG. 3

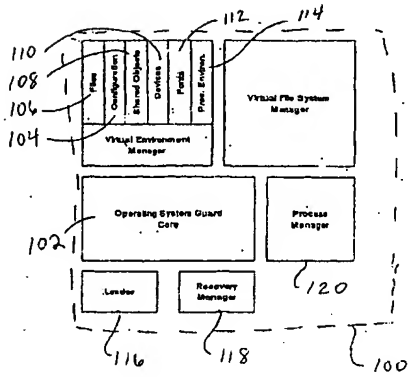

110
108
106
104
112
114
Files
Configuration
Shared Objects
Devices
Ports
Proc. Environ.
Virtual File System Manager
Virtual Environment Manager
Operating System Guard Core
Process Manager
Loader
Recovery Manager
102
120
116
118
100

FIG. 4

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 02/15378

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/445 G06F9/44 G06F15/163 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No
X	US 5 752 005 A (JONES CLAY LAMOYNE) 12 May 1998 (1998-05-12) abstract column 2, line 35 - line 60	1-17
X	US 6 026 402 A (VOSSEN JOSEPH K ET AL) 15 February 2000 (2000-02-15) column 2, line 24 - line 30 column 4, line 57 - line 67 column 5, line 31 - line 38 abstract	1-17
A	MARK RUSSINOVICH AND BRYCE COGSWELL: "Windows NT System-Call Hooking" DR. DOBB'S JOURNAL, January 1997 (1997-01), page 42,44,46,82 XP001096512 the whole document	4-6

-/-

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 September 2002	30/09/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Müller, T

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 02/15378

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 00 46685 A (AHN JAI WAN ;SONG DONG HO (KR); SOFTONNET CO LTD (KR)) 10 August 2000 (2000-08-10) abstract page 7, line 8 - line 11 page 7, line 30 - line 36 page 11, line 12 - line 20 figure 5A	9-14
A	US 6 023 721 A (CUMMINGS CHRISTOPHER R) 8 February 2000 (2000-02-08) abstract column 2, line 20 - line 55 column 3, line 44 - line 62 column 4, line 35 - line 60	13
A	EDOUARD BUGNION, SCOTT DEVINE AND MENDEL ROSENBLUM: "Disco: Running Commodity Operating Software on Scalable Multiprocessors" PROCEEDINGS OF THE 16TH SYMPOSIUM ON OPERATING SYSTEMS PRINCPLES (SOPS), 'Online! October 1997 (1997-10), pages 1-14, XP002211769 Saint-Malo, France Retrieved from the Internet: <URL:ftp://www-flash.stanford.edu/pub/hive /SOSP97-disco.ps> 'retrieved on 2002-08-30! page 1, left-hand column, line 39 - line 45 page 2, right-hand column, line 1 - line 10 page 6, right-hand column, line 25 - line 32	15-17
A	ROGUE WAVE SOFTWARE: "Copy on Write" WEBSITE OF ROGUE WAVE SOFTWARE, 'Online! 1996, XP002211770 Retrieved from the Internet: <URL:http://www.tacc.utexas.edu/resources/ user_guides/crayc++tools/toolsug/cop_3018. htm> 'retrieved on 2002-08-30! the whole document	15-17

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 02/15378

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 5752005	A	12-05-1998	NONE		
US 6026402	A	15-02-2000	NONE		
WO 0046685	A	10-08-2000	AU	2463700 A	25-08-2000
			CN	1354857 T	19-06-2002
			EP	1163599 A1	19-12-2001
			WO	0046685 A1	10-08-2000
US 6023721	A	08-02-2000	NONE		

Form PCT/ISA/210 (patent family annex) (July 1992)

フロントページの続き

（特許庁注：以下のものは登録商標）
Ｍａｃｉｎｔｏｓｈ

【要約の続き】
クス、システムデバイスドライバ、フォント、レジストリ、その他の構成項目、ファイル、環境変数が含まれる。
【選択図】図１